# Redactable vs. Sanitizable Signatures

Kai Samelin, Henrich C. Pöhls,
Joachim Posegga and Hermann de Meer
{ks,hp,jp}@sec.uni-passau.de, demeer@uni-passau.de

Institute of IT-Security and Security-Law (ISL), University of Passau, Germany

Technical Report, Number MIP-1208
Department of Informatics and Mathematics
University of Passau, Germany
December 2012

# Redactable vs. Sanitizable Signatures

Kai Samelin[1]*, Henrich C. Pöhls[2]**, Joachim Posegga[2], Hermann de Meer[1]

[1] Chair of Computer Networks and Computer Communication
[2] Chair of IT-Security
Institute of IT-Security and Security Law (ISL), University of Passau, Germany
{ks,hp,jp}@sec.uni-passau.de, demeer@uni-passau.de

**Abstract.** Malleable signature schemes allow altering signed data in a controlled way while keeping the signature verifiable trusting the signer's key. Several constructions exists. They can be grouped in two different categories: (1) redactable signatures ($\mathcal{RSS}$) and (2) sanitizable signatures ($\mathcal{SSS}$). $\mathcal{RSS}$s allow for removing blocks of a signed document, while $\mathcal{SSS}$s offer the possibility to change all admissible blocks to arbitrary strings. This paper shows that $\mathcal{SSS}$s with a strenghted security definition can be transformed into $\mathcal{RSS}$s with a weakened privacy definition. A transformation from a $\mathcal{RSS}$ into a $\mathcal{SSS}$ is not possible, even if we assume accountability for $\mathcal{RSS}$. In particular, no unforgeable $\mathcal{RSS}$ can be transformed into a $\mathcal{SSS}$. This work provides the first rigorous proof that $\mathcal{RSS}$s and $\mathcal{SSS}$s are two different concepts.

**Keywords:** Redactable Signatures, Sanitizable Signatures, Privacy, Malleable Signatures

## 1 Introduction

Standard digital signature schemes like RSA-PSS [5, 35] become invalid on any change to the signed data protected. However, this also prohibits a third party from changing this data in a controlled way. Applications, where such a controlled is crucial, include secure routing [2] or the anonymization of medical data [24]. It is also of paramount inmportance that the modification of the signed data requires no interaction between the sanitizer, i.e., the party who changes the signed data, and the original signer. This addresses constellations where the original signer is not reachable anymore, e.g., in case of death, or in the case where the signer may be reachable, but must not know which data is passed to other third parties. This may happen, if personal data like gender, age or place of birth is involved. A suitable approach to this so-called "digital document

*The research leading to these results was supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project (http://www.secbit.de) and the European Community's Seventh Framework Programme through the EINS Network of Excellence (grant agreement no. [288021]).

**Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project

sanitization problem" [30] are malleable signature schemes. Malleable signature schemes allow for certain controlled changes to signed data such that the resulting changed message's authenticity is still verifiable. Let $m = (m[1], \dots, m[\ell])$, where $\ell \in \mathbb{N}^+$, be a string $m$ split up into $\ell$ parts we refer to as *blocks*. A redactable signature scheme ($\mathcal{RSS}$) allows *everyone* to remove blocks $m[i]$ from $m$. In particular, a redaction of the block $m[i]$ leaves a blinded message $m'$ without $m[i]$, i.e., $m' = (\dots, m[i-1], \perp, m[i+1], \dots)$. If $\perp$ is visible has a major impact on the security model of the $\mathcal{RSS}$. It also requires that the third party can derive a signature $\sigma'$ which verifies for $m'$. On the other hand, a sanitizable signature scheme ($\mathcal{SSS}$) allows that a sanitizer, which has its own secret key, can change the *admissible* blocks, defined by the signer, into arbitrary strings $m[i]' \in \{0,1\}^*$. Hence, the sanitizer can generate a verifiable message/signature pair $(m', \sigma')$, where $\sigma'$ is the corresponding signature and $m' = (\dots, m[i]', \dots)$. Obviously, the derived signatures still need to induce trust, i.e., it must be verifyable that all changes were endorsed by the signer. On the first sight, both approaches aim for the same goal, i.e., sanitizing signed data. However, $\mathcal{SSS}$s only allow the **alteration** of blocks, while $\mathcal{RSS}$s only allow the **removal of complete** blocks. Moreover, $\mathcal{SSS}$s require an additional key pair for sanitization, while $\mathcal{RSS}$s allow for public redactions, i.e., no additional key pair is needed.

**Motivation.** Sanitizing digitally signed data becomes more and more important as digital signatures are deployed in business settings and for timestamping or archiving purposes [33, 39]. However, standard unforgeable digital signature do not allow any later modification to the data signed. Hence, we need to find solutions to the "digital document sanitization problem" [30]. Current provably secure solutions only focus on one specific type of malleable signature scheme, i.e., $\mathcal{RSS}$s *or* $\mathcal{SSS}$s, even though they see them as related work. Hence, the question arises how the relation between both types of malleable signature is. This paper addresses this gap and proves the minimal set of security definitions required to transform a $\mathcal{SSS}$ into a $\mathcal{RSS}$. We also prove that no transform can result in a fully private $\mathcal{RSS}$. This shows, that **every** existing transform is not secure, as the security models are not sufficient. Hence, we conclude that $\mathcal{RSS}$s and $\mathcal{SSS}$s must be combined to achieve a maximum amount of flexibility and security.

### 1.1 Contribution

This paper rigorously shows that $\mathcal{RSS}$s and $\mathcal{SSS}$s are completely different concepts, with one notable exception: weakening the privacy definition of $\mathcal{RSS}$s, while altering the security definitions of $\mathcal{SSS}$s, a $\mathcal{SSS}$ can be transformed into a $\mathcal{RSS}$. We provide a general algorithm for the transform and show that an $\mathcal{SSS}$ that is

- strongly private,
- weakly immutable and
- weakly blockwise non-interactive publicly accountable

can be transformed into a weakly private $\mathcal{RSS}$. We prove that this is the minimal set of assumptions required. *Strongly private* for $\mathcal{SSS}$s requires, that even if the sanitizing key is known, no statements about the original message can be made. *Weak blockwise non-interactive public accountability* prohibits a sanitizer from accusing the signer for a specific block. A *weakly immutable* $\mathcal{SSS}$ prohibits an adversary knowing the secret sanitizing key from altering blocks not designated to be sanitized. However, we also prove that the resulting $\mathcal{RSS}$s are only weakly private. In a *weakly private* $\mathcal{RSS}$, a third party can see where the message has been redacted, i.e., it sees the position of the change, but cannot make any additional statements, i.e., cannot revert the redaction.

We introduce the required formal security model in Sect. 2. Moreover, we show that no unforgeable $\mathcal{RSS}$ can be transformed into a $\mathcal{SSS}$. This results in the fact that $\mathcal{RSS}$s and $\mathcal{SSS}$s must be combined to achieve more flexibility in sanitizing signed data. Hence, our results rule out every existing transformations, as the existing security models are not suitable. We consider this the main contribution of this paper.

**State-of-the-Art.** $\mathcal{SSS}$s have been introduced by *Ateniese* et al. [2] at ESORICS '05. *Brzuska* et al. later formalized the most used security properties [7]. These have been later extended for unlinkability [9] and (blockwise) non-interactive public accountability [10]. Moreover, several extensions like limiting-to-values [11, 22, 31], trapdoor $\mathcal{SSS}$s [13, 41] and multi-sanitizer environments [8, 12] have been considered. Currently, the only work considering $\mathcal{SSS}$s and data-structures more complex than lists appeared in [31].

On the other hand, $\mathcal{RSS}$s have been introduced in [21], and in a slightly different way in [38]. Based on their work, many additional research appeared. Several cover more complex data-structures like trees [6, 24, 34, 31, 36] and graphs [26]. The standard security properties of $\mathcal{RSS}$s have been formalized in [6, 14, 32, 36]. How to force the signer to commit to a given message has been shown in [33], while *Ahn* et al. introduce the notion of statistically unlinkable $\mathcal{RSS}$s [1]. Even stronger privacy notions have been introduced in [3]. However, the scheme by *Ahn* et al. only achieves the less common notion of selective unforgeability [1]. There exists many additional work on $\mathcal{RSS}$s. However, most of the schemes are not private, e.g., [18–20, 27, 29, 40]. Hence, a verifier can make statements about the original message $m$, which contradicts the intention of a $\mathcal{RSS}$ [6].

Combinations of both approaches appeared in [18–20]. However, as already pointed out by *Samelin* et al., their schemes do not preserve privacy [37]. No other work considering combinations or relations of $\mathcal{SSS}$s and $\mathcal{RSS}$s is known to the authors.

### 1.2 Outline

The rest of the paper is structured as follows. In Sect. 2, we give the required preliminaries to understand our results. This section also introduces the new notions of strong privacy, weak immutability and weak blockwise non-interactive public accountability for $\mathcal{SSS}$s. It also provides the new definition of weak privacy for $\mathcal{RSS}$s. A general transform showing that a $\mathcal{SSS}$ with strong privacy, weak immutability and weak blockwise non-interactive public accountability can be transformed into an $\mathcal{RSS}$ with weak privacy is given in Sect. 3. Based on the preliminaries, we give formal proofs of the relation between $\mathcal{SSS}$s and $\mathcal{RSS}$s in Sect. 4. Our resulting scheme is subject to further modifications in Sect. 5. We conclude our work in Sect. 6. Additional formal proofs of security are found in the appendix.

## 2 Preliminaries

For a message $m = (m[1], \ldots, m[\ell])$, we call $m[i] \in \{0,1\}^*$ a *block*, while "," denotes a uniquely reversible concatenation of blocks or strings. The symbol $\perp \notin \{0,1\}^*$ denotes an error or an exception. For a visible redaction, we use the symbol $\Box \notin \{0,1\}^*$, $\Box \neq \perp$.

### 2.1 Sanitizable Signatures

The used nomenclature is adapted from *Brzuska* et al. [7], which is also true for the following definition:

**Definition 1 (Sanitizable Signature Scheme).** *Any $\mathcal{SSS}$ consists of at least seven efficient, i.e., $\mathcal{PPT}$ algorithms. In particular, let $\mathcal{SSS} := (\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$, such that:*

**Key Generation.** *There are two key generation algorithms, one for the signer and one for the sanitizer. Both create a pair of keys, a private key and the corresponding public key, based on the security parameter $\lambda$:*

$$(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^\lambda)$$
$$(\mathrm{pk}_{san}, \mathrm{sk}_{san}) \leftarrow \mathsf{KGen}_{san}(1^\lambda)$$

**Signing.** *The $\mathsf{Sign}$ algorithm takes as input the security parameter $\lambda$, a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, the secret key $\mathrm{sk}_{sig}$ of the signer, the public key $\mathrm{pk}_{san}$ of the sanitizer, as well as a description* ADM *of the admissibly modifiable blocks, where* ADM *contains the number $\ell$ of blocks in $m$, as well the indices of the modifiable blocks. It outputs the message $m$ and a signature $\sigma$ (or $\perp$, indicating an error):*

$$(m, \sigma) \leftarrow \mathsf{Sign}(1^\lambda, m, \mathrm{sk}_{sig}, \mathrm{pk}_{san}, \textsc{adm})$$

**Sanitizing.** *Algorithm* $\mathsf{Sanit}$ *takes a message* $m = (m[1], \dots, m[\ell])$, $m[i] \in \{0,1\}^*$, *the security parameter* $\lambda$, *a signature* $\sigma$, *the public key* $\mathrm{pk}_{sig}$ *of the signer and the secret key* $\mathrm{sk}_{san}$ *of the sanitizer. It modifies the message* $m$ *according to the modification instruction* $\textsc{mod}$, *which contains pairs* $(i, m[i]')$ *for those blocks that shall be modified.* $\mathsf{Sanit}$ *calculates a new signature* $\sigma'$ *for the modified message* $m' \leftarrow \textsc{mod}(m)$. *Then* $\mathsf{Sanit}$ *outputs* $m'$ *and* $\sigma'$ *(or possibly* $\perp$ *in case of an error).*

$$(m', \sigma') \leftarrow \mathsf{Sanit}(1^\lambda, m, \textsc{mod}, \sigma, \mathrm{pk}_{sig}, \mathrm{sk}_{san})$$

**Verification.** *The* $\mathsf{Verify}$ *algorithm outputs a decision* $d \in \{\mathtt{true}, \mathtt{false}\}$ *verifying the correctness of a signature* $\sigma$ *for a message* $m = (m[1], \dots, m[\ell])$, $m[i] \in \{0,1\}^*$ *with respect to the public keys* $\mathrm{pk}_{sig}$ *and* $\mathrm{pk}_{san}$ *and the security parameter* $\lambda$*:*

$$d \leftarrow \mathsf{Verify}(1^\lambda, m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san})$$

**Proof.** *The* $\mathsf{Proof}$ *algorithm takes as input the security parameter, the secret signing key* $\mathrm{sk}_{sig}$, *a message* $m = (m[1], \dots, m[\ell])$, $m[i] \in \{0,1\}^*$ *and a signature* $\sigma$ *as well a set of (polynomially many) additional message/signature pairs* $\{(m_i, \sigma_i) \mid i \in \mathbb{N}^+\}$ *and the public key* $\mathrm{pk}_{san}$. *It outputs a string* $\pi \in \{0,1\}^*$ *(or* $\perp$*, indicating an error):*

$$\pi \leftarrow \mathsf{Proof}(1^\lambda, \mathrm{sk}_{sig}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}^+\}, \mathrm{pk}_{san})$$

**Judge.** *Algorithm* $\mathsf{Judge}$ *takes as input the security parameter, a message* $m = (m[1], \dots, m[\ell])$, $m[i] \in \{0,1\}^*$ *and a valid signature* $\sigma$, *the public keys of the parties and a proof* $\pi$. *It outputs a decision* $d \in \{\mathtt{Sig}, \mathtt{San}, \perp\}$ *indicating whether the message/signature pair has been created by the signer or the sanitizer (or* $\perp$*, indicating an error):*

$$d \leftarrow \mathsf{Judge}(1^\lambda, m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, \pi)$$

To have an algorithm actually able to derive the accountable party for a specific block $m[i]$, *Brzuska* et al. introduced the additional algorithm $\mathsf{Detect}$ [10]. The algorithm $\mathsf{Detect}$ is not part of the original $\mathcal{SSS}$ description by *Ateniese* et al., since it is not required for the purpose of a $\mathcal{SSS}$ [2, 7]. However, it is required to later define (weak) blockwise non-interactive public accountability (See Def. 6).

**Definition 2 (($\mathcal{SSS}$) Detect).** *On input of the security parameter* $\lambda$, *a message/signature pair* $(m, \sigma)$, *the corresponding public keys* $\mathrm{pk}_{sig}$ *and* $\mathrm{pk}_{san}$, *and the block index* $1 \leq i \leq \ell$, $\mathsf{Detect}$ *outputs the accountable party (*$\mathtt{San}$ *or* $\mathtt{Sig}$*) for block* $i$ *(or* $\perp$*, indicating an error):*

$$d \leftarrow \mathsf{Detect}(1^\lambda, m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, i), d \in \{\mathtt{San}, \mathtt{Sig}, \perp\}$$

We require the usual correctness properties to hold. In particular, all genuinely signed or sanitized messages are accepted, while every genuinely created proof by the signer leads the judge to decide in favor of the signer. For a formal definition of correctness, refer to [7, 10]. It is also required by every $\mathcal{SSS}$ that ADM is always correctly recoverable from any valid message/signature pair $(m, \sigma)$. Jumping ahead, we want to emphasize that a $\mathcal{SSS}$ with weak non-interactive public accountability has an empty Proof algorithm and a Judge that detects any sanitization, based on any proof $\pi \in \{0,1\}^* \cup \perp$. We give formal definitions of the security properties in a game-based manner after introducing $\mathcal{RSS}$s.

### 2.2 Redactable Signatures

The following notation is derived from [6] and [37].

**Definition 3 (Redactable Signature Schemes).** *A $\mathcal{RSS}$ consists of four efficient algorithms. In particular, let $\mathcal{RSS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact})$ such that:*

**KeyGen.** *The algorithm KeyGen outputs the public key* pk *and private key* sk *of the signer, where $\lambda$ is the security parameter:*

$$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$$

**Sign.** *The algorithm Sign gets as input the security parameter $\lambda$, the secret key* sk *and the message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$:*

$$(m, \sigma) \leftarrow \mathsf{Sign}(1^\lambda, \mathrm{sk}, m)$$

**Verify.** *The algorithm Verify outputs a decision $d \in \{\mathtt{true}, \mathtt{false}\}$, indicating the correctness of the signature $\sigma$, w.r.t.* pk*, protecting $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$:*

$$d \leftarrow \mathsf{Verify}(1^\lambda, \mathrm{pk}, m, \sigma)$$

**Redact.** *The algorithm Redact takes as input the message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, the public key* pk *of the signer, a valid signature $\sigma$, a list of indizes* MOD *of blocks to be redacted and the security parameter $\lambda$. It returns a modified message $m' \leftarrow \mathrm{MOD}(m)$ (or $\perp$, indicating an error):*

$$(m', \sigma') \leftarrow \mathsf{Redact}(1^\lambda, \mathrm{pk}, m, \sigma, \mathrm{MOD})$$

*We denote the transitive closure of $m$ as $span_{\models}(m)$. This set contains all messages derivable from $m$ w.r.t. Redact*

As for $\mathcal{SSS}$s, the correctness properties for $\mathcal{RSS}$s are required to hold as well. Thus, every genuinely signed or redacted message must verify. Refer to [6] for a formal definition of correctness.

**Security Models.** In this section, the needed security properties and models required for our proofs are introduced. They are derived from [7, 17, 37]. The requirement that ADM is always correctly reconstructable is captured within the unforgeability and immutability definitions. Note, following [7], a $\mathcal{SSS}$ must at least be unforgeable, immutable, accountable and private to be meaningful. Hence, we assume that all used $\mathcal{SSS}$s fulfill these four fundamental security requirements; if these requirements are not met, the construction is not considered a $\mathcal{SSS}$ and the results of this paper may differ. For the following definitions of security properties, we merge descriptions of $\mathcal{SSS}$s and $\mathcal{RSS}$s where possible, allowing to see the parallels.

**Definition 4 (($\mathcal{RSS}$) Unforgeability).** *No one should be able to compute a valid signature on a message not previously queried without having access to any private keys [6]. That is, even if an outsider can request signatures on different documents, it remains hard to forge a signature for a document not previously signed. This is analogous to the standard unforgeability requirement for standard signature schemes [16], except that it excludes valid redactions from the set of forgeries. We say that a $\mathcal{RSS}$ is unforgeable, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 1 returns* 1*, is negligible (as a function of $\lambda$).*

**Experiment** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)$
    let $i = 1, \ldots, q$ index the queries
  return 1, if
    $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$ and
    $\forall i, 1 \leq i \leq q : m^* \notin \mathrm{span}_{\models}(m_i)$

**Fig. 1.** Unforgeability for $\mathcal{RSS}$s

**Definition 5 (($\mathcal{SSS}$) Unforgeability).** *As before, no one should be able to generate valid signatures on new documents not queried before without having access to any private keys. For $\mathcal{SSS}$s, we also have to take the sanitization and proof oracles into account [7]. Again, this is analogous to the standard unforgeability requirement for standard signature schemes [16], except that it excludes valid sanitizations from the set of forgeries. We say that a $\mathcal{SSS}$ is unforgeable, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 2 returns* 1*, is negligible (as a function of $\lambda$).*

**Definition 6 (($\mathcal{SSS}$) Weak Blockwise Non-Interactive Public Accountability).** *A sanitizable signature scheme $\mathcal{SSS}$ is* weakly non-interactive publicly

**Experiment** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda)$
$(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KGen}_{\mathrm{sig}}(1^{\lambda})$
$(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KGen}_{\mathrm{san}}(1^{\lambda})$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, \cdots)\mathsf{Proof}(\cdot, sk_{\mathrm{sig}}, \cdots)}_{\mathsf{Sanit}(\cdots, sk_{\mathrm{san}})}(pk_{\mathrm{sig}}, pk_{\mathrm{san}})$
return 1, if
$\mathsf{Verify}(m^*, \sigma^*, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}$ and
$(\textsc{adm}_i \neq \textsc{adm}^*$ or
$m^*$ has not been returned by an oracle)

**Fig. 2.** Unforgeability for $\mathcal{SSS}$s

**Experiment** $\mathsf{WBlockPubAcc}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda)$
$(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KGen}_{\mathrm{sig}}(1^{\lambda})$
$(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KGen}_{\mathrm{san}}(1^{\lambda})$
$(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, pk_{\mathrm{san}}, \cdot)}_{\mathsf{Proof}(\cdot, sk_{\mathrm{sig}}, \cdot, \cdot, pk_{\mathrm{san}})}(pk_{\mathrm{san}}, sk_{\mathrm{san}}, pk_{\mathrm{sig}})$
Let $(m_i, \textsc{adm}_i, pk_{\mathrm{san}})$ and $\sigma_i$ for $i = 1, \ldots, k$
be the queries/answers to/from $\mathcal{O}^{\mathsf{Sign}}$
return 1, if
$\mathsf{Verify}(1^{\lambda}, m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*) = \mathtt{true}$, and
for all $m_i$ with $pk_{\mathrm{san},i} = pk^*$, $\exists q$, s.t.
$\mathsf{Detect}(1^{\lambda}, m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*, q) = \mathtt{Sig}$
and $(q, m_i[q]) \in \textsc{mod}_i$.
return 0

**Fig. 3.** Weak Blockwise Non-Interactive Public Accountability for $\mathcal{SSS}$s

accountable, *if* $\mathsf{Proof} = \perp$, *and if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment given in Fig. 3 returns* 1 *is negligible (as a function of* $\lambda$*). The basic idea is that an adversary, i.e., the sanitizer, has to be able to make the* $\mathsf{Detect}$ *algorithm, accuse the signer, if it did not sign the specific block. Please note, the sanitizer key is not generated by the adversary. An example for a weakly blockwise non-interactive publicly accountable* $\mathcal{SSS}$ *is the scheme introduced by Brzuska et al. [10]. Note, in our definition the signer is* ***not*** *considered adversarial, contrary to Brzuska et al. [10]. Hence, we do not need to consider the case where the signer accuses the sanitizer, as done in [10]. We explain the reasons for our adversary model after the introduction of all required security properties.*

**Definition 7 (($\mathcal{SSS}$) Standard Privacy).** *No one should be able to gain any knowledge about sanitized parts without having access to them [7]. This is similar to the standard indistinguishability notion for encryption schemes [15]. The basic idea is that the oracle either signs and sanitizes the first message or the second, while the resulting message must be the same for each input. The adversary must not be able to decide which input message was used. We say that a* $\mathcal{SSS}$ *is* *strongly private, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 4 returns* 1*, is negligibly close to* $\frac{1}{2}$ *(as a function of* $\lambda$*).*

**Experiment** $\mathsf{Privacy}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^{\lambda})$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^{\lambda})$
$b \leftarrow \{0, 1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(sk_{\text{sig}},\cdots),\mathsf{Proof}(sk_{\text{sig}},\cdots)}_{\mathsf{LoRSanit}(\ldots,sk_{\text{sig}},sk_{\text{san}},b),\mathsf{Sanit}(\cdots,sk_{\text{san}})}(pk_{\text{sig}}, pk_{\text{san}})$
where oracle $\mathsf{LoRSanit}$ on input of:
$m_0, \text{MOD}_0, m_1, \text{MOD}_1, \text{ADM}$
if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
let $(m, \sigma) \leftarrow \mathsf{Sign}(m_b, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
return $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}_b, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
return 1, if $a = b$

**Fig. 4.** Standard Privacy for $\mathcal{SSS}$s

**Definition 8 (($\mathcal{SSS}$) Strong Privacy).** *The basic idea remains the same: no one should be able to gain any knowledge about sanitized parts without having access to them, with one exception: the adversary is given the secret key* $\text{sk}_{san}$ *of the sanitizer. Hence, the adversary must not be able to decide which input message was used. We say that a* $\mathcal{SSS}$ *for documents is private, if for any efficient (PPT) adversary* $\mathcal{A}$ *the probability that the game depicted in Fig. 5 returns* 1*, is negligibly close to* $\frac{1}{2}$ *(as a function of* $\lambda$*). This notion extends the definition of standard privacy [7] to also account for parties knowing the secret sanitizer key* $\text{sk}_{san}$*. Examples for strongly private* $\mathcal{SSS}$*s are the scheme introduced by Brzuska et al. [10], as both of their schemes are information-theoretically private.*

The privacy definition in [7] only considers outsiders as adversarial. However, we require that even insiders, i.e., sanitizers, are not able to win the game. This is similar to the game given in [12], with the notable exception that the key $sk_{\text{san}}$ is **not** generated by the adversary, only known to it. We explain the need for this alteration after the next definitions.

**Definition 9 (($\mathcal{RSS}$) Weak Privacy).** *In a weakly private* $\mathcal{RSS}$*, a third party can derive which parts of a message have been redacted without gathering more information, as redacted blocks are replaced with* $\perp$*. The basic idea is that the oracle either signs and sanitizes the first message or the second. As before, the resulting redacted message* $m'$ *must be the same for both inputs, with one additional exception: the length of both inputs must be the same, while* $\perp$ *is considered part of the message. For strong privacy, this constraint is not required. We say that*

**Experiment** $\mathsf{SPrivacy}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^{\lambda})$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^{\lambda})$
$b \leftarrow \{0, 1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(sk_{\text{sig}},\cdots),\mathsf{Proof}(sk_{\text{sig}},\cdots)}_{\mathsf{LoRSanit}(\ldots,sk_{\text{sig}},sk_{\text{san}},b)}(pk_{\text{sig}}, pk_{\text{san}}, sk_{\text{san}})$
where oracle $\mathsf{LoRSanit}$ on input of:
$m_0, \text{MOD}_0, m_1, \text{MOD}_1, \text{ADM}$
if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
let $(m, \sigma) \leftarrow \mathsf{Sign}(m_b, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
return $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}_b, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
return 1, if $a = b$:

**Fig. 5.** Strong Privacy for $\mathcal{SSS}$s

**Experiment** $\mathsf{WPrivacy}_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$b \xleftarrow{\$} \{0, 1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}_{\mathsf{LoRRedact}(\ldots,sk,b)}(pk)$
where oracle $\mathsf{LoRRedact}$
for input $m_0, m_1, \text{MOD}_0, \text{MOD}_1$:
if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
Note, **visible** redacted parts are denoted $\Box$,
which are considered part of the message
$(m, \sigma) \leftarrow \mathsf{Sign}(sk, m_b)$
return $(m', \sigma') \leftarrow \mathsf{Redact}(pk, m, \sigma, \text{MOD}_b)$.
return 1, if $b = d$

**Fig. 6.** Weak Privacy for $\mathcal{RSS}$s

*a $\mathcal{RSS}$ for documents is **weakly** private, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 6 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of $\lambda$). We want to emphasize, that Lim et al. define weak privacy in a different manner, i.e., they prohibit access to the signing oracle [27]. Our definition allows such adaptive queries. Summarized, weak privacy just makes statements about blocks, not the complete message. See [24] for possible attacks. Weakly private schemes, following our definition, are, e.g., [18–20, 24, 25, 27]. In their schemes, the adversary is able to pinpoint the indices of the redacted blocks, as $\perp$ is visible.*

**Definition 10 (($\mathcal{RSS}$) Strong Privacy).** *This definition is similar to weak privacy. However, redacted parts are not considered part of the message. We say that a $\mathcal{RSS}$ for documents is strongly private, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 7 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of $\lambda$). This is the standard definition of privacy [6, 37].*

**Experiment** $\mathsf{SPrivacy}_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$b \xleftarrow{\$} \{0, 1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}_{\mathsf{LoRRedact}(\ldots,sk,b)}(pk)$
where oracle $\mathsf{LoRRedact}$
for input $m_0, m_1, \text{MOD}_0, \text{MOD}_1$:
if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
redacted blocks are **not**
considered part of the message
$(m, \sigma) \leftarrow \mathsf{Sign}(sk, m_b)$
return $(m', \sigma') \leftarrow \mathsf{Redact}(pk, m, \sigma, \text{MOD}_b)$.
return 1, if $b = d$

**Fig. 7.** Strong Privacy for $\mathcal{RSS}$s

**Definition 11 (($\mathcal{RSS}$) Transparency).** *Another well-known security definition is transparency [6, 32]. Interpreting the formal definition, which is depicted in Fig. 8, transparency is the anonymity of the signer, i.e., a third party cannot decide whether a given message/signature pair $(m, \sigma)$ originates from the signer or the sanitizer. A redactable signature scheme $\mathcal{RSS}$ is transparent, if for any efficient algorithm $\mathcal{A}$ the probability that the experiment given in Fig. 8 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of $\lambda$). The basic idea is that an adversary has access to an oracle which either signs and then sanitizes the message or vice versa. Following the argumention and proofs given in [10], we can derive that no weakly blockwise non-interactive publicly accountable scheme can be transparent [10].*

**Experiment** $\mathsf{Transparency}_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$b \xleftarrow{\$} \{0, 1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{Sign/Redact}(\ldots,sk,b)}(pk)$
where oracle $\mathsf{Sign/Redact}$ for input $m, \text{MOD}$:
if $\text{MOD}(m) \notin \text{span}_{\models}(m)$, return $\perp$
if $b = 0$: $(m, \sigma) \leftarrow \mathsf{Sign}(sk, m)$,
$(m', \sigma') \leftarrow \mathsf{Redact}(pk, \sigma, m, \text{MOD})$
if $b = 1$: $m' \leftarrow \text{MOD}(m)$
$(m', \sigma') \leftarrow \mathsf{Sign}(sk, m')$,
finally return $(m', \sigma')$.
return 1, if $b = d$

**Fig. 8.** Transparency for $\mathcal{RSS}$s

**Experiment** $\mathsf{Immutability}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda)$
$(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$(m^*, \sigma^*, pk^*_{\mathrm{san}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, \cdot, \cdot), \mathsf{Proof}(sk_{\mathrm{sig}}, \cdots)}(pk_{\mathrm{sig}})$
return 1, if:
$\mathsf{Verify}(m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*_{\mathrm{san}}) = \mathtt{true}$ and
$\exists i : m^*[j_i] \neq m_i[j_i]$ for some $j_i \notin \mathrm{ADM}_i$ or
$pk^*_{\mathrm{san}} \neq pk_{\mathrm{san}, i}$
shorter messages are padded with $\perp$ or
$\mathrm{ADM}_i \neq \mathrm{ADM}^*$

**Fig. 9.** Immutability for $\mathcal{SSS}$s

**Definition 12 (($\mathcal{SSS}$) Immutability).** *A sanitizable signature scheme $\mathcal{SSS}$ is immutable, iff for any efficient algorithm $\mathcal{A}$ the probability that the experiment given in Fig. 9 returns 1 is negligible (as a function of $\lambda$) [7]. The basic idea is that an adversary generating the sanitizer key must be able to sanitize a block not designated to be sanitized. Note, the sanitizer key is created by the adversary. In other words, immutability is the unforgeability requirement for the sanitizer.*

**Definition 13 (($\mathcal{SSS}$) Weak Immutability).** *A sanitizable signature scheme $\mathcal{SSS}$ is weakly immutable, iff for any efficient algorithm $\mathcal{A}$ the probability that the experiment given in Fig. 10 returns 1 is negligible (as a function of $\lambda$). The basic idea is that an adversary knowing the sanitizer key must be able to sanitize a block not designated to be sanitized. Note, the sanitizer key is not created by the sanitizer, only known.*

**Experiment** $\mathsf{WImmutability}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda)$
$(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$(m^*, \sigma^*, pk^*_{\mathrm{san}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, pk_{\mathrm{san}}, \cdot), \mathsf{Proof}(sk_{\mathrm{sig}}, \cdots, pk_{\mathrm{san}})}(pk_{\mathrm{sig}}, pk_{\mathrm{san}}, sk_{\mathrm{san}})$
return 1, if:
$\mathsf{Verify}(m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*_{\mathrm{san}}) = \mathtt{true}$ and
$\exists i : m^*[j_i] \neq m_i[j_i]$ for some $j_i \notin \mathrm{ADM}_i$
shorter messages are padded with $\perp$ or
$\mathrm{ADM}_i \neq \mathrm{ADM}^*$

**Fig. 10.** Weak Immutability for $\mathcal{SSS}$

Interestingly, weak immutability is enough for our construction to be unforgeable, while for a $\mathcal{SSS}$ used in the normal way, this definition is obviously not suitable at all.

### 2.3 Implications and Separations.

Let us formulate our first theorems:

**Theorem 1.** *Every $\mathcal{RSS}$ which is strongly private, is also weakly private.*

*Proof.* The game for strong privacy is less restrictive for the adversary than weak privacy. Hence, weak privacy is implied by strong privacy.

**Theorem 2.** *There exists a $\mathcal{RSS}$ which is weakly private, but not strongly private.*

*Proof.* See [18–20, 27] for examples. Additionally, we show that our scheme is not strongly private, i.e., only weakly, in App. A.

**Theorem 3.** *Every $\mathcal{SSS}$ which is immutable is also weakly immutable.*

*Proof.* The game for immutability is less restrictive for the adversary than weak immutability. Hence, immutability implies weak immutability.

**Theorem 4.** *There exists a $\mathcal{SSS}$ which is private, but not strongly private.*

*Proof.* We show this theorem by modifying an arbitrary existing strongly private $\mathcal{SSS}$. Let $\mathcal{SSS} = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be an arbitrary private $\mathcal{SSS}$. We alter the scheme as follows:

- $\mathsf{KGen}'_{\mathrm{sig}} := \mathsf{KGen}_{\mathrm{sig}}$, i.e., the key generation algorithm for the signer remains unchanged.
- $\mathsf{KGen}'_{\mathrm{san}} := \mathsf{KGen}_{\mathrm{san}}$, while an additional key pair for a IND-CCA2-secure encryption scheme [4] $\mathcal{ENC}$ is generated.
- $\mathsf{Sign}'$ is the same as $\mathsf{Sign}$ with one exception; it appends the encryption $e$ of a digest of original message to the final signature, i.e., $\sigma' = (\sigma, e)$, where $e \leftarrow \mathcal{ENC}(pk_{\mathrm{san}}, \mathcal{H}(m))$ and $\mathcal{H}$ some standard cryptographic hash-function.
- $\mathsf{Sanit}'$ is the same as $\mathsf{Sanit}$ with one exception; it first removes the encrypted digest from the signature, while it appends it to the resulting signature.
- $\mathsf{Verify}'$ is the same as $\mathsf{Verify}$ with one exception; it removes the encrypted digest from the signature before verifying.
- $\mathsf{Proof}'$ and $\mathsf{Judge}'$ work essentially the same as their original counterparts, while cutting of $e$ from the signature before proceeding.

Clearly, a sanitizer, holding the corresponding secret key for $\mathcal{ENC}$, can distinguish between **messages** generated by the signer and the sanitizer with overwhelming probability using the information contained in the **signature** $\sigma$. Without $sk_{\mathrm{san}}$, this information remains hidden due to the IND-CCA2 encryption.

### 2.4 Definition of a Secure $\mathcal{RSS}$ and a Secure $\mathcal{SSS}$.

We want to explicitly emphasize that accountability, as defined for $\mathcal{SSS}$s in [7], has not been defined for $\mathcal{RSS}$s yet, as $\mathsf{Redact}$ is a public algorithm. Hence, no secret sanitizer key(s) are required to allow any modifications. To circumvent this inconsistency, we utilize a standard $\mathcal{SSS}$ and let the signer generate the sanitizer key $sk_{\text{san}}$, attaching it to the signature, i.e., $\sigma' = (\sigma, sk_{\text{san}})$. If any alteration without $sk_{\text{san}}$ would be possible, the underlying $\mathcal{SSS}$ is obviously forgeable. As we have defined that this is a non-secure $\mathcal{SSS}$, we omit this case. By doing so, the secret $sk_{\text{san}}$ becomes public knowledge and can be used by every party. This makes redacting a public operation. Hence, the secret key $sk_{\text{san}}$ for the sanitization becomes known to every party, including the signer to remain in the model defined for $\mathcal{RSS}$s. This is the reason for our modifications of the existing security notions. We require these, on first sight very unnatural, restrictions to stay consistent with the standard model of $\mathcal{SSS}$s as formalized in [7]. Moreover, the signer is generally **not** considered an adversarial entity in $\mathcal{RSS}$s [33] . If other notions or adversary models are used, the results may obviously differ. In Sect. 4, we show that any $\mathcal{SSS}$ which only achieves standard privacy is not enough to construct a weakly private $\mathcal{RSS}$ and additional impossibility results. We show, without giving formal definitions, how one can derive an accountable $\mathcal{RSS}$ with explicitly denoted sanitizers in Sect. 5. How to formalize accountability for $\mathcal{RSS}$s is an open question and is not answered within this paper.

As we aim to transform a $\mathcal{SSS}$ into a $\mathcal{RSS}$, which by definition allows public redactions, we stick with the second approach, i.e., we require that $\mathsf{KGen}_{\text{san}}$ is called by the signer and the resulting $sk_{\text{san}}$ is distributed by being publicly reconstructable from the signature. Since we have also dropped the requirement of an $\mathcal{SSS}$ to be accountable, we require weak blockwise non-interactive public accountability only for detecting (admissible) changes of blocks, twisting up the meaning its name implies.[1] Vice versa, i.e., for the proof that no unforgeable $\mathcal{RSS}$ can be transformed into an unforgeable $\mathcal{SSS}$, we neither require any additional security definitions nor any multi-sanitizer environments as introduced in [12]. In particular, we only need the unforgeability requirements of $\mathcal{RSS}$s. Obviously, unforgeability is the most basic requirement, essential for **every** meaningful cryptographic construction.

We conclude this section with two final definitions:

**Definition 14 (Secure $\mathcal{SSS}$).** *We call a $\mathcal{SSS}$ secure, if, and only if, it is strongly private, weakly immutable, unforgeable and weakly blockwise non-interactive public accountable.*

**Definition 15 (Secure $\mathcal{RSS}$).** *We call a $\mathcal{RSS}$ secure, if, and only if, it is weakly private and unforgeable.*

[1] To be more precise: [10] uses this feature to derive if the message has been sanitized: if this is the case, the sanitizer must be responsible and is therefore accountable

## 3 Generic Transformation of an $\mathcal{SSS}$ into an $\mathcal{RSS}$

This section presents the generic transform. In particular, we give a generic algorithm to transform any unforgeable, strongly private, and weakly blockwise non-interactive publicly accountable $\mathcal{SSS}$ into an unforgeable and weakly private $\mathcal{RSS}$.

**Outline.** The basic idea of our transform is that every party, including the signer, is allowed to alter **all** given blocks. The verification procedure accepts sanitized blocks, if, and only if, the altered blocks are $\square$. $\square$ is then be treated as a redacted block. Hence, redaction is altering a given block to a special symbol. As we have to defined that a $\mathcal{SSS}$ only allows strings $m[i] \in \{0,1\}^*$, we need to define $\square := \emptyset$ and

$$m[i] \mapsto \begin{cases} 0 & \text{if } m[i] = \emptyset \\ m[i]+1 & \text{else} \end{cases}$$

to codify the additional symbol $\perp$, where $\emptyset$ expresses the empty string. Hence, we remain in the model defined. Moreover, this is where weak blockwise non-public interactive public accountability comes in: the changes to *each* block need to be detectable to allow a meaningful result, as a $\mathcal{SSS}$ allows arbitrary alterations. As $\perp$ is still visible, the resulting scheme can only be weakly private, as statements about the original message $m$ can be made, contradicting our definition of strong privacy. Moreover, as a $\mathcal{RSS}$ allows that every party can redact blocks, we require that the sanitizing key $sk_{\mathrm{san}}$ is known to every party, including the signer. Therefore, we need a strongly private $\mathcal{SSS}$ to achieve our definition of weak privacy for our $\mathcal{RSS}$, as we prove in Sect. 4.

**The Transform.** In this section, we give a generic algorithm to actually perform the transform.

**Construction 1** *Let $\mathcal{SSS} := (\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge}, \mathsf{Detect})$ be a secure $\mathcal{SSS}$. Let the message space contain no $\square$ symbol. Define the redactable signature scheme $\mathcal{RSS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact})$ as follows:*

*Key Generation: Algorithm $\mathsf{KeyGen}$ generates on input of the security parameter $\lambda$, a key pair $(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) \leftarrow \mathcal{SSS}.\mathsf{KGen}_{sig}(1^\lambda)$ of the $\mathcal{SSS}$, and also a sanitizer key pair $(\mathrm{pk}_{san}, \mathrm{sk}_{san}) \leftarrow \mathcal{SSS}.\mathsf{KGen}_{san}(1^\lambda)$. It returns $(\mathrm{sk}, \mathrm{pk}) = (\mathrm{sk}_{sig}, (\mathrm{sk}_{san}, \mathrm{pk}_{san}, \mathrm{pk}_{sig}))$*

*Signing: Algorithm $\mathcal{RSS}.\mathsf{Sign}$ on input $m \in \{0,1\}^*, \mathrm{sk}, \mathrm{pk}$, sets $\mathrm{ADM} = (1, \ldots, \ell)$ and computes $\sigma_s \leftarrow \mathcal{SSS}.\mathsf{Sign}(1^\lambda, m, \mathrm{sk}_{sig}, \mathrm{pk}_{san}, \mathrm{ADM})$. It outputs: $(m, \sigma)$, where $\sigma = (\mathrm{sk}_{san}, \sigma_s)$*

*Redacting: Algorithm $\mathcal{RSS}.\mathsf{Redact}$ on input message $m$, modification instructions $\mathrm{MOD}$, a signature $\sigma = (\mathrm{sk}_{san}, \sigma_s)$, keys $\mathrm{pk}_{sig}$ and $\mathrm{pk}_{san}$ first checks if*

*$\sigma$ is a valid signature for $m$ under the given public keys using $\mathcal{RSS}.\mathsf{Verify}$. If not, it stops outputting $\perp$. Afterwards, it sets $\textsc{mod}' = \{(i, \square) \mid i \in \textsc{mod}\}$. In particular, it generates a modification description for the $\mathcal{SSS}$ which sets block with index $i \in \textsc{mod}$ to $\square$. Finally, it computes $(m', \sigma'_s) \leftarrow \mathcal{SSS}.\mathsf{Sanit}(1^\lambda, m, \textsc{mod}', \sigma_s, \mathrm{pk}_{sig}, \mathrm{sk}_{san})$ and outputs $(m', \sigma')$, where $\sigma' = (\mathrm{sk}_{san}, \sigma'_s)$*

*Verification: Algorithm $\mathcal{RSS}.\mathsf{Verify}$ on input a message $m \in \{0,1\}^*$, a signature $\sigma = (\mathrm{sk}_{san}, \sigma_s)$ and public keys $\mathrm{pk}_{sig}$, $\mathrm{pk}_{san}$ first checks that $\textsc{adm} = (1, \dots, \ell)$ and that $\sigma_s$ is a valid signature for $m$ under the given public keys using $\mathcal{SSS}.\mathsf{Verify}$. If not, it returns `false`. Afterwards, for each $i$ for which $\mathcal{SSS}.\mathsf{Detect}(1^\lambda, m, \sigma_s, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, i)$ returns `San`, it checks that $m[i] = \square$. If not, it returns `false`. Else, it returns `true`. Note, one may also check if the given sanitizer key $\mathrm{sk}_{san}$ is correct*

Obviously, the secret sanitizer $sk_{\mathrm{san}}$ must be known to alter a block. That is the reason why it must be part of the signature to allow *public* redactions, as required by $\mathcal{RSS}$s. Every redaction becomes visible as a message block containing the special symbol $\square$.

**Theorem 5 (Our Construction is Secure).** *If the utilized $\mathcal{SSS}$ is weakly blockwise non-interactive publicly accountable, weakly immutable and strongly private, the resulting $\mathcal{RSS}$ is weakly private and unforgeable, i.e., secure.*

*Proof.* Th. 5 is proven in App. A.

**Theorem 6 (Our Construction is not Strongly Private).** *Our construction is only weakly private, but not strongly private.*

*Proof.* Due to Th. 5, we already know that our scheme is weakly private. Hence, it remains to show that it is not strongly private. As a redaction leaves a **visible** special symbol, i.e., $\square$, an adversary can win the **strong** privacy game in the following way: Generate two messages $m_0, m_1$, where $m_1 = (m_0, 1)$. Hence, $\ell_0 < \ell_1$, while $m_0$ is a prefix of $m_1$. Afterwards, it requests that $m_1[\ell_1]$ is redacted, i.e., $\textsc{mod}_1 = (\ell_1)$ and $\textsc{mod}_0 = ()$. Hence, if the oracle chooses $b = 0$, it will output $m_2 = m_0$ and for $b = 1$, $m_2 = (m_1, \square)$. Obviously, the adversary can always win the game, as $(m_1, \square) \neq m_0$.

Note, in the strong privacy game,$\perp$ is not considered part of the message $m$. Hence, the scheme cannot be strongly private.

As $\mathcal{RSS}$s allow every block to be removed, we require that $\textsc{adm} = (1, \dots, \ell)$. This rules out cases where a signer prohibits alterations of blocks, i.e., in our case, what we require for redaction. We show in Sect. 5 how this constraint can be transformed into the useful notion of consecutive disclosure control. In particular, there exists security models for $\mathcal{RSS}$s, where prohibiting redactions is allowed and claimed to be useful, e.g., in [28, 37].

## 4 Minimum Requirements for a $\mathcal{SSS}$ to be Transformed

In this section, we show that standard private $\mathcal{SSS}$s are not enough to build weakly private $\mathcal{RSS}$s. Moreover, we prove that weak blockwise non-interactive public accountability is required to build an unforgeable $\mathcal{RSS}$. To formally express this intuitive goals we need the next theorems:

**Theorem 7 (Any non strongly private $\mathcal{SSS}$, results in a non-weakly private $\mathcal{RSS}$).** *If the transformed $\mathcal{SSS}$ is only private, but not strongly private, the resulting $\mathcal{RSS}$ is not weakly private.*

*Proof.* Let $\mathcal{A}$ be an adversary winning the strong privacy game as defined in Fig. 5. We can then construct an adversary $\mathcal{B}$, which wins the weak privacy game as defined in Fig. 6, using $\mathcal{A}$ as a black-box in the following way:

1. The challenger generates $sk_{\text{sig}}, sk_{\text{san}}, pk_{\text{sig}}, pk_{\text{san}}$ and passes all but $sk_{\text{sig}}$ to $\mathcal{B}$
2. $\mathcal{B}$ passes all received keys to $\mathcal{A}$. Note, $sk_{\text{san}}$ is required to for public redactions
3. $\mathcal{B}$ simulates the signing oracle using the oracle provided by the challenger
4. Eventually, $\mathcal{A}$ returns its guess $b^*$
5. $\mathcal{B}$ outputs $b^*$ as its own guess

Following the definitions, the success probability of $\mathcal{A}$ is non-negligible, the success probability of $\mathcal{B}$ is non-negligible. In particular, the success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$. This proves the theorem.

**Theorem 8 (No Transform can Result in a Strongly Private $\mathcal{RSS}$).** *There exists no algorithm which transforms a weakly immutable $\mathcal{SSS}$ into a strongly private $\mathcal{RSS}$.*

*Proof.* Once again, every meaningful $\mathcal{SSS}$ must be immutable, which implies weak immutability due to Th. 3. Hence, we do not make any statements about schemes not weakly immutable. We show that any transform $\mathcal{T}$ achieving this property uses a $\mathcal{SSS}'$ which is not weakly immutable. Also, our definition of a $\mathcal{SSS}$ requires that ADM is **always** recoverable. Let $\mathcal{RSS}'$ denote the resulting $\mathcal{RSS}$. We can then derive an algorithm which uses $\mathcal{RSS}'$ to break the weak immutability requirement of the underlying $\mathcal{SSS}$ in the following way:

1. The challenger generates the two key pairs of the $\mathcal{SSS}$. It passes all keys but $sk_{\text{sig}}$ to $\mathcal{A}$
2. $\mathcal{A}$ transforms the $\mathcal{SSS}$ into $\mathcal{RSS}'$ given the transform $\mathcal{T}$
3. $\mathcal{A}$ calls the oracle $\mathcal{SSS}.\mathsf{Sign}$ with a message $m = (1, 2)$ and simulates $\mathcal{RSS}'.\mathsf{Sign}$
4. $\mathcal{A}$ calls $\mathcal{RSS}'.\mathsf{Redact}$ with MOD $= (1)$

5. If the resulting signature $\sigma$ does not verify, abort
6. Output the resulting signature $\sigma_{\mathcal{SSS}}$ of the underlying $\mathcal{SSS}$

As $\ell_m \neq \ell_{\text{MOD}(m)}$, $(\text{MOD}(m), \sigma_{\mathcal{SSS}})$ breaks the weak immutability requirement of the $\mathcal{SSS}$, as ADM is altered, contradicting our definition of a secure $\mathcal{SSS}$. Moreover, as hiding redacted parts of a message is essential for strong privacy, no algorithm exists, which transforms a weakly immutable $\mathcal{SSS}$ into a strongly private $\mathcal{RSS}$, as ADM needs to be correctly recoverable. This proves the theorem. Note, we can give a concrete counterexample as we only use required behavior.

**Theorem 9 (Weak Blockwise Non-Interactive Accountability is Required for any Transform $\mathcal{T}$).** *For any transformation algorithm $\mathcal{T}$, the utilized $\mathcal{SSS}$ must be weakly blockwise non-interactive publicly accountable to result in an unforgeable $\mathcal{RSS}$.*

*Proof.* Let $\mathcal{RSS}'$ be the resulting $\mathcal{RSS}$. Perform the following steps to show that the resulting $\mathcal{RSS}$ is forgeable. In particular, let $\mathcal{A}$ winning the weak blockwise non-interactive accountability game, which is used by $\mathcal{B}$ to break the unforgeability of the resulting $\mathcal{RSS}$.

1. The challenger generates the two key pairs of the $\mathcal{SSS}$. It passes all keys but $sk_{\text{sig}}$ to $\mathcal{B}$
2. $\mathcal{B}$ forwards all received keys to $\mathcal{A}$
3. Any calls to the signing oracle by $\mathcal{A}$ are answered genuinely by $\mathcal{B}$ using its own signing oracle
4. Eventually, $\mathcal{A}$ returns a tuple $(m, \sigma_{\mathcal{SSS}})$ to $\mathcal{B}$
5. If the resulting signature does not verify or does not win the weak blockwise non-interactive accountability game, $\mathcal{A}$ and therefore also $\mathcal{B}$ abort
6. $\mathcal{A}$ transforms the $\mathcal{SSS}$ into $\mathcal{RSS}'$ given the transform $\mathcal{T}$
7. If the resulting signature does not verify, $\mathcal{B}$ aborts
8. $\mathcal{B}$ outputs the resulting $(m', \sigma_{\mathcal{RSS}'})$ of the resulting $\mathcal{RSS}'$

Following our definition in Fig. 3, $(m', \sigma_{\mathcal{RSS}'})$ breaks the unforgeability requirement of the $\mathcal{RSS}$, as there exists a block which has not been signed by the signer. Moreover, the success probabilities are equal.

**Theorem 10 (No Unforgeable $\mathcal{RSS}$ can be Transformed into a $\mathcal{SSS}$).** *There exists no transform $\mathcal{T}$, which converts an unforgeable $\mathcal{RSS}$ into a $\mathcal{SSS}$.*

*Proof.* Let $\mathcal{SSS}'$ be the resulting $\mathcal{SSS}$. Now perform the following steps to extract a valid forgery of the underlying $\mathcal{RSS}$:

1. The challenger generates a key pair for a $\mathcal{RSS}$. It passes $pk$ to $\mathcal{A}$.

2. $\mathcal{A}$ transforms $\mathcal{RSS}$ into $\mathcal{SSS}'$ given the transform $\mathcal{T}$
3. $\mathcal{A}$ calls the oracle $\mathcal{RSS}.\mathsf{Sign}$ with a message $m = (1, 2)$ and simulates $\mathcal{SSS}'.\mathsf{Sign}$ with $\textsc{adm} = (1)$
4. $\mathcal{A}$ calls $\mathcal{SSS}'.\mathsf{Sanit}$ with $\textsc{mod} = (1, a)$
5. If the resulting signature does not verify, abort
6. Output the resulting signature $\sigma_{\mathcal{RSS}}$ of the underlying $\mathcal{RSS}$

As $(a, 2) \notin \mathrm{span}_{\models}(m)$, $((a, 2), \sigma_{\mathcal{RSS}})$ is a valid forgery of the underlying $\mathcal{RSS}$. Note, this concrete counterexample is possible, as only required behavior is used.

## 5 Extensions

This section introduces additional modifications to our transform, which results in new properties not considered yet in previous works.

### 5.1 Consecutive Disclosure Control

We require $\textsc{adm} = (1, \ldots, \ell)$, i.e. all message blocks are admissible to change. However, as already pointed out by *Miyazaki* et al. and *Samelin* et al., prohibiting consecutive redactions is a very useful feature [28, 29, 37]. In their case, the signer or an intermediate recipient is able to prohibit consecutive redaction. With our method, we achieve something different but related: we can prohibit that any consecutive party is able to prohibit sanitization. In particular, if $\textsc{adm} \neq (1, \ldots, \ell)$, i.e., $\overline{\textsc{adm}} = (1, \ldots, \ell) \setminus \textsc{adm}$, a consecutive redaction is limited to blocks which are part of $\textsc{adm}$. Obviously, this feature relies on the weak immutability property of the underlying $\mathcal{SSS}$. Moreover, an intermediate recipient is able to remove the secret key $sk_{\mathrm{san}}$ from the signature to prohibit any further redactions. To disallow this possibility, the sanitizing key must be signed and not just appended to the signature. If a different sanitizing key for each block is used, this allows a blockwise consecutive redaction control. We leave it as open work to formally define these properties.

### 5.2 Restricting to Sanitizers and Accountability

All $\mathcal{RSS}$s allow everyone to redact blocks. To limit redaction to explicitly denoted sanitizers, the signature $\sigma$ is extended to hold an additional signature $\sigma_2$. Let $\sigma_2 \leftarrow \mathrm{SIGN}(sk, \mathcal{CH}(pk_{\mathcal{CH}}, m))$, where $\mathcal{CH}$ is a chameleon hash [23]. The values required to calculate $\mathcal{CH}$ need to be delivered with $\sigma$. Hence, only sanitizers who possess the secret key $sk_{\mathcal{CH}}$ for $\mathcal{CH}$ can sanitize the message $m$ without invalidating the signature. This can be enriched further to achieve sanitizer and signer accountability [7]: $\mathcal{CH}$ could be replaced with a tag-based chameleon-hash $\mathcal{CH}_{TAG}$, i.e., the construction of *Brzuska* et al. [7]. This idea has already been proposed by *Samelin* et al. [36, 37], but can also be applied for our scheme. However, a formal definition is still missing and thus as open work.

## 6 Conclusion

This paper presents how a $\mathcal{SSS}$ can be transformed into a $\mathcal{RSS}$, if the corresponding security models are slightly adjusted. We gave a generic transform and proved the resulting $\mathcal{RSS}$ to be weakly private and unforgeable. Hence, all existing transforms are not suitable, as their security model is not strong enough to give sufficient privacy guarantees. We introduced the minimal set of security properties for an $\mathcal{SSS}$ that are required to yield a secure $\mathcal{RSS}$. These strong notions have not been considered in previous work. Moreover, we give a rigorously argument that no $\mathcal{RSS}$ can be transformed into an unforgeable $\mathcal{SSS}$. This implies, that $\mathcal{SSS}$s and $\mathcal{RSS}$s are completely different concepts.

It remains an open question, how to formally define accountability for $\mathcal{RSS}$s and how $\mathcal{RSS}$s and $\mathcal{SSS}$s can be combined to yield a more flexible, yet fully private, sanitizable and redactable signature scheme. It also remains open, if unlinkable $\mathcal{SSS}$s result in unlinkable $\mathcal{RSS}$s.

## References

1. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096, 2011. http://eprint.iacr.org.
2. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS: Proceedings of the 10th European Symposium on Research in Computer Security*, pages 159–177. Springer-Verlag, 2005.
3. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT*, pages 367–385, 2012.
4. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO*, pages 26–45, 1998.
5. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994.
6. C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ACNS'10, pages 87–104. Springer, 2010.
7. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer, 2009.

8. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. In *Proc. of BIOSIG*, volume 155 of *LNI*, pages 117–128. GI, 2009.
9. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of Sanitizable Signatures. In *Public Key Cryptography*, pages 444–461, 2010.
10. C. Brzuska, H. C. Pöhls, and K. Samelin. Non-Interactive Public Accountability for Sanitizable Signatures. In *EuroPKI 9th European Workshop*, volume ???? of *LNCS*, pages ??–?? Springer-Verlag, 2012.
11. S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.
12. S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AFRICACRYPT*, pages 35–52, 2012.
13. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.
14. E.-C. Chang, C. L. Lim, and J. Xu. Short Redactable Signatures Using Random Trees. In *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, CT-RSA '09, pages 133–147, Berlin, Heidelberg, 2009. Springer-Verlag.
15. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
16. S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17:281–308, 1988.
17. J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, volume 6584 of *Lecture Notes in Computer Science*, pages 300–317. Springer Berlin / Heidelberg, 2011.
18. S. Haber, Y. Hatano, Y. Honda, W. G. Horne, K. Miyazaki, T. Sander, S. Tezoku, and D. Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS*, pages 353–362, 2008.
19. T. Izu, M. Izumi, N. Kunihiro, and K. Ohta. Yet another sanitizable and deletable signatures. In *AINA Workshops*, pages 574–579, 2011.
20. T. Izu, N. Kunihiro, K. Ohta, M. Sano, and M. Takenaka. Sanitizable and deletable signature. In Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors, *Information Security Applications*, volume 5379 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2009.
21. R. Johnson, D. Molnar, D. Song, and D.Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Security Conference - Cryptographers Track*, pages 244–262. Springer, Feb. 2002.
22. M. Klonowski and A. Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.
23. H. Krawczyk and T. Rabin. Chameleon Hashing and Signatures. In *Symposium on Network and Distributed Systems Security*, pages 143–154, 2000.
24. A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. In *Proc. of PVLDB 2008*, New Zealand, 2008. ACM.
25. A. Kundu and E. Bertino. CERIAS Tech Report 2009-1 Leakage-Free Integrity Assurance for Tree Data Structures, 2009.
26. A. Kundu and E. Bertino. How to authenticate graphs without leaking. In *EDBT*, pages 609–620, 2010.
27. S. Lim, E. Lee, and C.-M. Park. A short redactable signature scheme using pairing. *Security and Communication Networks*, 5(5):523–534, 2012.

28. K. Miyazaki, G. Hanaoka, and H. Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ASIACCS '06, pages 343–354, New York, NY, USA, 2006. ACM.
29. K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka, and H. Imai. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246, 2005.
30. K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshiura. Digital documents sanitizing problem. Technical report, IEICE, 2003.
31. H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *Applied Cryptography and Network Security, 9th International Conference*, volume 6715 of *LNCS*, pages 166–182. Springer-Verlag, 2011.
32. H. C. Pöhls, K. Samelin, J. Posegga, and H. de Meer. Length-hiding redactable signatures from one-way accumulators in $\mathcal{O}(n)$ (mip-1201). Technical report, University of Passau, 4 2012.
33. H. C. Pöhls, K. Samelin, J. Posegga, and H. de Meer. Transparent mergeable redactable signatures with signer commitment and applications (mip-1206). Technical report, University of Passau, 8 2012.
34. H. C. Pöhls, Kai Samelin, H. de Meer, and J. Posegga. Flexible redactable signature schemes for trees - extended security model and construction. In *SECRYPT*, pages 113–125, 2012.
35. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.
36. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. On Structural Signatures for Tree Data Structures. In *Applied Cryptography and Network Security, 10th International Conference*, volume 7341 of *LNCS*, pages 171–187. Springer-Verlag, 2012.
37. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer-Verlag, 2012.
38. R. Steinfeld and L. Bull. Content extraction signatures. In *Information Security and Cryptology - ICISC 2001: 4th International Conference*. Springer Berlin / Heidelberg, 2002.
39. K. W. Tan and R. H. Deng. Applying Sanitizable Signature to Web-Service-Enabled Business Processes: Going Beyond Integrity Protection. In *ICWS*, pages 67–74, 2009.
40. Z.-Y. Wu, C.-W. Hsueh, C.-Y. Tsai, F. Lai, H.-C. Lee, and Y. Chung. Redactable Signatures for Signed CDA Documents. *Journal of Medical Systems*, pages 1–14, December 2010.
41. D. H. Yum, J. W. Seo, and P. J. Lee. Trapdoor sanitizable signatures made easy. In *Proceedings of the 8th international conference on Applied cryptography and network security*, ACNS'10, pages 53–68, Berlin, Heidelberg, 2010. Springer-Verlag.


## A Proofs

*Proof.* This will prove Th. 5. We have to show that the resulting $\mathcal{RSS}$ is unforgeable and weakly private. We prove each property on its own.

I) **Unforgeability.** Let $\mathcal{A}$ be an algorithm breaking the unforgeability of the resulting $\mathcal{RSS}$. We can then construct an algorithm $\mathcal{B}$ which breaks the weak blockwise non-interactive publicly accountability of the utilized $\mathcal{SSS}$. To do so, $\mathcal{B}$ simulates $\mathcal{A}$'s environment in the following way:

1. $\mathcal{B}$ receives the following keys: $pk_{\text{san}}, sk_{\text{san}}, pk_{\text{sig}}$ and forwards them to $\mathcal{A}$
2. For every query to the signing oracle, $\mathcal{B}$ forwards the query to its own signing oracle and therefore is able to perfectly simulate the signing oracle for $\mathcal{A}$
3. Eventually, $\mathcal{A}$ outputs a tuple $(m^*, \sigma^*)$
4. If $(m^*, \sigma^*)$ does not verify or is trivial, abort

$\mathcal{B}$ outputs $(m^*, \sigma^*)$ as its own forgery. Following the definition of unforgeability, $m$ cannot be derived from any queried message to the signature oracle, with the notable exception of $m[i] = \perp$ for any index $i$. Hence, there must exist at least one block $m[i] \neq \perp$, which has not been signed by the signer. Following our verification algorithm, the accepting verification requires that $\texttt{Sig} = \mathsf{Detect}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}, i)$. Hence, $(m^*, \sigma^*)$ breaks the weak blockwise non-interactive publicly accountability by outputting $(m^*, \sigma^*)$.

II) **Weak Privacy.** To show that our scheme is weakly private, we only need to show that an adversary $\mathcal{A}$ can derive information about the prior content of a contained block $m[i]$, as $\square$ is considered part of the resulting message $m'$ and all other modifications result in a forgeable $\mathcal{RSS}$. Let $\mathcal{A}$ winning the weak privacy game. We can then construct an adversary $\mathcal{B}$ which breaks the strong privacy game in the following way:

1. $\mathcal{B}$ receives the following keys: $pk_{\text{san}}, sk_{\text{san}}, pk_{\text{sig}}$ and forwards them to $\mathcal{A}$
2. For every query to the signing oracle, $\mathcal{B}$ forwards the query to its own signing oracle and therefore is able to perfectly simulate the signing oracle for $\mathcal{A}$
3. $\mathcal{B}$ also forwards any queries to its own $\mathsf{LoRSanit}$ oracle. It passes the answers to $\mathcal{A}$
4. Eventually, $\mathcal{A}$ outputs its guess $b^*$

$\mathcal{B}$ outputs $b^*$ as its own guess. As defined, the oracle requires that $\textsc{mod}_1(m_1) = \textsc{mod}(m_2)$, including any redacted blocks, i.e., $\square$. In particular, the messages are the same. Hence, the success probability of $\mathcal{B}$ is the same as $\mathcal{A}$'s. This proves the theorem.